Installation of Python on Linux

Using the Package Manager

1.  If you are using Ubuntu Linux, at the command prompt type:

$ sudo apt-get install python3

2.  If you are using Fedora, at the command prompt type:

$ sudo yum install python3

3.  The most recent version of Python 3 will be downloaded and installed. To verify the installation, type:

$ Python

The Python prompt (>>>) will appear.

From source

You can install Python from its source code if you want to really customize the binary by setting certain options or flags during the build process.

1. Download the source from <https://www.python.org/ftp/python/3.6.2/Python-3.6.2rc1.tgz>

2.  Type the following commands to extract and install Python from its source:

$sudo tar xzf python-3.6.2.tar.xz

$./configure

$sudo make install

3. To verify the installation, in the terminal type:

$ Python3

4. The Python prompt (>>>) will appear.

Installation of Python on Mac OS X

Using a wizard-based installer

1. Go to the Python website, https://www.python.org/ftp/python/3.6.2/python-3.6.2-macosx10.6.pkg and download the required version.

2. Run the downloaded file and follow the instructions in the installation wizard.

Using Homebrew

Homebrew is a package manager that lets you install, update, and uninstall packages from the command line on the Mac OS.

1. Homebrew depends on Apple’s Xcode package, so run the following command to install Xcode first:

$ xcode-select --install

2. Next, install Homebrew by following the instructions on their website, <https://brew.sh/>

3. After installing Homebrew, from the prompt in the terminal type the following command to install Python:

$ brew install python3

4. To verify the installation, type:

$ Python3

The Python prompt (>>>) will appear.

Installation of Python on other platforms

You can install Python for other platforms such as MS-DOS, OS2 or Solaris from the Python website,<https://www.python.org/download/other/>

Alternative Python implementations

The standard distribution of Python (available on python.org) is developed in C Language and is often calledCPython. This recommended standard is tweaked to optimize its performance for specific applications. These implementations combine ease of development of Python and rich libraries of other platforms such as .net or Java. Some such alternative implementations are:

 IronPython (Python running on .NET framework. Developed in C#)

 Jython (Developed in Java. Python running on the Java Virtual Machine. Capable of using rich Java library in Python program)

 PyPy (A [fast](http://speed.pypy.org/) python implementation with a JIT compiler)

 Stackless Python (Branch of CPython supporting microthreads)

 MicroPython (Python running on micro controllers)

Commercial distributions of Python

While community versions are open source and intended to use on as is basis, commercial versions of Python ensure guarantee of service and maintenance of the products. Also, when modules and packages are installed from repositories, it may lead to version conflict in a community version. However, commercial versions ensure that the distribution is self-contained and without compatibility issues. Some of the commercial distributions of Python include:

 ActiveState ActivePython (commercial and community versions, including scientific computing modules)

 pythonxy (Scientific-oriented Python Distribution based on Qt and Spyder)

 winpython (WinPython is a portable scientific Python distribution for Windows)

 Enthought Canopy (a commercial distribution for scientific computing)

 PyIMSL Studio (a commercial distribution for numerical analysis )

 [Anaconda Python](https://store.continuum.io/cshop/anaconda) (a full Python distribution for data management, analysis and visualization of large data sets)

Keywords in Python

|  |  |  |
| --- | --- | --- |
| 'False' | 'elif' | 'lambda' |
| 'None' | 'else' | 'nonlocal' |
| 'True' | 'except' | 'not' |
| 'and' | 'finally' | 'or' |
| 'as' | 'for' | 'pass' |
| 'assert' | 'from' | 'raise' |
| 'break' | 'global' | 'return' |
| 'class' | 'if' | 'try' |
| 'continue' | 'import' | 'while' |
| 'def' | 'in' | 'with' |
| 'del' | 'is' | 'yield' |

1. Which syntax element is used to define a code block? (A code block is a group of statements that, together, perform a task.)

(1 Mark)

1. Comment
2. Triple Quotes, """ """
3. Indent
4. Curly brackets, { }

Correct Answer : c

2. Is the following statement true or false?  
  
Python is a compiler-based language.

(2 Marks)

1. True
2. False

Correct Answer : b

3. Which of the following statements is False?

(3 marks)

1. Parentheses are mandatory with the print function in Python 3.x but not in Python 2.x.
2. The raw\_input() function has been deprecated (discontinued) in Python 3.x.
3. Integer objects are long by default in Python 3.x but not in Python 2.x in which a trailing L is required.
4. In Python 3.x, 5/2 will result in an output of 2.

Correct Answer : d

4. Given the following code in IDLE, what will you see after you type print(a,b,b) and press Enter?   
  
>>> a=2  
>>>b=3  
>>>print(a,b,b) 

(1 Mark)

1. 18
2. a b b
3. 2 3 3
4. Error

5. What will you see if you type 19-4\*5 and press Enter in IDLE?

(2 Marks)

1. 759375
2. -1
3. 75
4. Error

Correct Answer : b

6. What is the correct command for displaying all the Python keywords?

(3 marks)

1. >>> import keyword  
   >>> keyword.kwlist
2. >>> keyword.python
3. >>> import keyword.kwlist  
   >>> keyword.python
4. >>> import .keyword.kwlist

7. The result of 36%5 is:

(1 Mark)

1. 1
2. 7
3. 7.2
4. 1.75

8. What will you see if you type a = 12 \* 2 and press Enter in IDLE?

(2 Marks)

1. 24
2. 144
3. Nothing
4. Error

9. There are some syntax errors in the displayed program. Of the options displayed, select the option which is the correct version of this program.  
  
#The purpose of this  
program is to display some text. #  
  
print ("Hello World")   
print ("Welcome to Internshala") #Author Internshala

(3 marks)

1. #The purpose of this  
   program is to display some text. #  
     
   print (Hello World)  
   print (Welcome to Internshala) #Author Internshala
2. #The purpose of this  
   #program is to display some text.  
     
   print ("Hello World")  
   print ("Welcome to Internshala") #Author Internshala
3. #  
   The purpose of this  
   program is to display some text.   
   #  
     
   print ("Hello World")  
   print ("Welcome to Internshala") #Author Internshala

10. Is the following statement true or false?  
  
Dynamic Typing is a feature of Python.

(1 Mark)

1. True
2. False

1. Which syntax element is used to define a code block? (A code block is a group of statements that, together, perform a task.)

(1 Mark)

1. Comment
2. Triple Quotes, """ """
3. Indent
4. Curly brackets, { }

Correct Answer : c

2. Is the following statement true or false?  
  
Python is a compiler-based language.

(2 Marks)

1. True
2. False

Correct Answer : b

3. Which of the following statements is False?

(3 marks)

1. Parentheses are mandatory with the print function in Python 3.x but not in Python 2.x.
2. The raw\_input() function has been deprecated (discontinued) in Python 3.x.
3. Integer objects are long by default in Python 3.x but not in Python 2.x in which a trailing L is required.
4. In Python 3.x, 5/2 will result in an output of 2.

Correct Answer : d

4. Given the following code in IDLE, what will you see after you type print(a,b,b) and press Enter?   
  
>>> a=2  
>>>b=3  
>>>print(a,b,b) 

(1 Mark)

1. 18
2. a b b
3. 2 3 3
4. Error

5. What will you see if you type 19-4\*5 and press Enter in IDLE?

(2 Marks)

1. 759375
2. -1
3. 75
4. Error

Correct Answer : b

6. What is the correct command for displaying all the Python keywords?

(3 marks)

1. >>> import keyword  
   >>> keyword.kwlist
2. >>> keyword.python
3. >>> import keyword.kwlist  
   >>> keyword.python
4. >>> import .keyword.kwlist

7. The result of 36%5 is:

(1 Mark)

1. 1
2. 7
3. 7.2
4. 1.75

8. What will you see if you type a = 12 \* 2 and press Enter in IDLE?

(2 Marks)

1. 24
2. 144
3. Nothing
4. Error

9. There are some syntax errors in the displayed program. Of the options displayed, select the option which is the correct version of this program.  
  
#The purpose of this  
program is to display some text. #  
  
print ("Hello World")   
print ("Welcome to Internshala") #Author Internshala

(3 marks)

1. #The purpose of this  
   program is to display some text. #  
     
   print (Hello World)  
   print (Welcome to Internshala) #Author Internshala
2. #The purpose of this  
   #program is to display some text.  
     
   print ("Hello World")  
   print ("Welcome to Internshala") #Author Internshala
3. #  
   The purpose of this  
   program is to display some text.   
   #  
     
   print ("Hello World")  
   print ("Welcome to Internshala") #Author Internshala

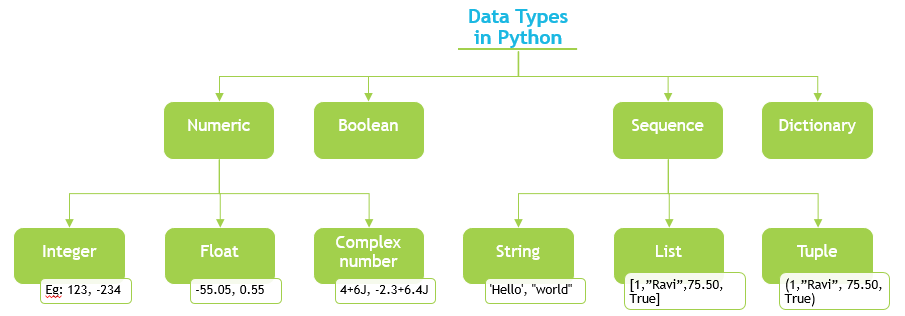
10. Is the following statement true or false?  
  
Dynamic Typing is a feature of Python.

(1 Mark)

1. True
2. False

Data Types in Python

The built-in data types in Python are displayed below.



|  |  |  |
| --- | --- | --- |
| Numeric |  | Any representation of data which has numeric value. Python identifies three types of numbers – integer, float and complex number. |
|  | Integer | Positive and negative whole numbers.  Examples: 1234, -234, 0x46 (hexadecimal number), 0O123 (octal number))  Note: In C and related programming languages such as Python, a hexadecimal number is prefixed with 0x and an octal number is prefixed with 0O. |
| Float | Real numbers with a floating point representation in which the fractional component is denoted by a decimal or scientific notation  Examples: -55.550, 0.005, 1.32E10 (scientific notation)) |
| Complex number | A number with a real and imaginary component is represented as a + bj inPython where a and b are floats and  j = √-1  ​​​Examples: 4+6j, -2.3+6.4j  Note: The common mathematical representation of a complex number uses a +bi with i being the imaginary part. But in electronics j is used because i already represent current and the next letter after i is j. |
| Boolean |  | Any representation of data which has two values denoted by True and False. |
| Sequence |  | An ordered collection of similar or different data types. The built-in Sequence data types in Python are – String, List and Tuple. |
|  | String | A collection of one or more characters put in single, double or triple quotes.  Examples: ‘Hello’, "Hello", "'Hello'", """Hello""" |
|  | List | An ordered collection of one or more data items, not necessarily of same type, put in square brackets.  Examples:  [1,"Ravi",75.50, True] |
|  | Tuple | An ordered collection of one or more data items, not necessarily of same type put in parentheses. The contents of a tuple cannot be modified – it is immutable - after the tuple is created.  Examples: (1,"Ravi", 75.50, True)  Note: Refer to the Helper Text to learn more about mutability. |
| Dictionary |  | An unordered collection of data in key:value pair form. Collection of such pairs is enclosed in curly brackets.  Example:  {1:"Superman", 2:"Wonder Woman", 3:"Thor", 4: "Hulk", 5:"Black Widow"} |

PREV

NEXT

* Helper Text
* Forum

Mutable and Immutable Objects

When a program is run, data objects in the program are stored in the computer’s memory for processing. While some of these objects can be modified at that memory location, other data objects can’t be modified once they are stored in the memory. *The property of whether or not data objects can be modified in the same memory location where they are stored is called*mutability*.*

We can check the mutability of an object by checking its memory location before and after it is modified. If the memory location remains the same when the data object is modified, it means it is mutable.

To check the memory location of where a data object is stored, we use the function, id(). Consider the following example (you can try this yourself in IDLE):

|  |  |
| --- | --- |
| >>> a=[5, 10, 15] | Assigning values to the list a. |
|  |  |
| >>> id(a) | Using the function id() to get the memory location of a. |
|  |  |
| 1906292064 | The ID of the memory location where a is stored. |
|  |  |
| >>> a[1]=20 | Replacing the second item in the list,10 with a new item, 20. |
|  |  |
| >>> print(a) | Using the print() function to verify the new value of a. |
|  |  |
| [5, 20, 15] | Verified that the value of a has changed. |
|  |  |
| >>> id(a) | Using the function id() to get the memory location of a. |
|  |  |
| 1906292064 | The ID of the memory location where a is stored. |

Notice that the memory location has not changed as the ID remains (1906292064) remains the same before and after the variable is modified. This indicates that the list is mutable, i.e., it can be modified at the same memory location where it is stored. Now, let us check if a tuple is mutable in Python.

|  |  |
| --- | --- |
| >>> b=(5, 10, 15) | Assigning values to the tuple b. |
|  |  |
| >>> b[1]=20 | Replacing the second item in the list, 10 with a new item, 20. |
|  |  |
| Traceback (most recent call last):    File "<pyshell#1>", line 1, in <module>      b[1]=20  TypeError: 'tuple' object does not support item assignment | Error explaining that a tuple does not support a modification in the items – i.e, it is immutable. |

You can verify the mutability of each of the data types in IDLE.

Immutable: numeric, string, and tuple  
Mutable: list, dictionary

Next live chat: Mon (10:00 - 11:00 PM)

Formatting Strings using Escape Sequences

You can use two or more specially designated characters within a string to format a string or perform a command. These characters are called escape sequences. An escape sequence in Python starts with a backslash (\). For example, \n is an escape sequence in which the common meaning of the letter n is literally escaped and given an alternative meaning - a new line.

Displayed here are a few common escape sequences available in python. You can try these out in IDLE or the Python prompt from the windows command prompt.

|  |  |  |  |
| --- | --- | --- | --- |
| Escape sequence | Description | Example | Result |
| \n | Breaks the string into a new line | >>> print('I designed this rhyme to explain in due time\nAll I know') | I designed this rhyme to explain in due time All I know |
| \t | Adds a horizontal tab | >>> print('Time is a \tvaluable thing') | Time is a valuable thing |
| \\ | Prints a backslash | >>> print('Watch it fly by\\ as the pendulum swings') | Watch it fly by\ as the pendulum swings |
| \' | Prints a single quote | >>> print('It doesn\'t even matter how hard you try') | It doesn't even matter how hard you try |
| \" | Prints a double quote | >>> print('It is so \"unreal\"') | It is so "unreal" |
| \a | makes a sound like a bell | >>> print('\a') | Sound effect is heard |

[ : ] Fetches characters in the range specified by two index operands separated by a colon.  
If the first operand is omitted, the range starts at zero index.  
If the second operand is omitted, the range goes up to the end of the string.  
Note: The slice starts at the first index. The slice ends one index before the second index, that is at the value of the index - 1.   
  
Example

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| T | h | e |  | B | u | r | r | o | W |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

>>> a='The Burrow' #String a

>>> a[2:7] #Starting index = 2 = e, Ending index = 7-1 = r

'e Bur'

>>> a[:6] #Starting index = 0 = T, Ending index = 6-1 = u

'The Bu'

>>> a[5:] #Starting index = 5 = u, Ending index = end of string = w

'urrow'

>>>

[in] Returns true if a character exists in the given string.  
  
Example

>>> a='Harry watched Dumbledore striding up and down in front of him, and thought. He thought of his mother, his father and Sirius. He thought of Cedric Diggory.' #String a

>>> 'v' in a #Checks if the character 'v' is present in the string, a

False

>>> 'dig' in a #Checks if the characters 'dig' are present in the string, a

False

>>> 'Dig' in a #Note that this is case-sensitive. As 'Dig' is present in 'Diggory', this returns True.

True

>>>

[not in] Returns true if a character does not exist in the given string.  
  
Example

>>> a=''' “For HIM?” shouted Snape. “Expecto Patronum!"

From the tip of his wand burst the silver doe: She landed on the office floor, bounded once across the office, and soared out of the window. Dumbledore watched her fly away, and as her silvery glow faded he turned back to Snape, and his eyes were full of tears.

“After all this time?”

“Always,” said Snape.''' #Multi-line string, a

>>> 'v' not in a #Checks that the character 'v' is not present in the string, a

False

>>> 'Red' not in a #Checks that the characters 'Red' iare not present in the string, a

True

>>> 'red' notin a #This is case sensitive. Since 'red' is present in 'soared', this returns False.

False

>>>

Format specification symbols

|  |  |  |
| --- | --- | --- |
| Format Symbol | Conversion |  |
| %c | character |  |
| %s | string conversion via str() prior to formatting | Example 1  >>> balltype='basketball' >>> result='hit' >>> print('I wondered why the %s was getting bigger. Then it %s me.' % (balltype, result)) I wondered why the basketball was getting bigger. Then it hit me. >>>   Example 2  >>> print("%20s" % ('Internshala', ))          Internshala >>>   Example 3  >>> print("%-20s" % ('Internshala', )) Internshala          >>>   Example 4  >>> print("%.5s" % ('Internshala', )) Inter >>> |
| %i | signed decimal integer |  |
| %d | signed decimal integer | >>> match=12553 >>> site='eBay' >>> print("%s is so useless. I tried to look up lighters and all they had was %d matches." % (site, match)) eBay is so useless. I tried to look up lighters and all they had was 12553 matches. >>> |
| %u | unsigned decimal integer |  |
| %o | octal integer |  |
| %x / %X | hexadecimal integer (lowercase letters) |  |

The format() Method

With Python 3.0, the format() method has been introduced for handling complex string formatting more efficiently. This method of the built-in string class provides functionality for complex variable substitutions and value formatting. This new formatting technique is regarded as more elegant. The general syntax of format() method is:

string.format(var1, var2,...)

The string itself contains placeholders {} in which values of variables are successively inserted.

>>> name="Malhar"

>>> age=23

>>> percentage=55.5

>>> "my name is {} and my age is {} years".format(name, age)

'my name is Malhar and my age is 23 years'

>>>

You can also specify formatting symbols. Only change is using colon (:) instead of %. For example, instead of %s use {:s} and instead of %d use (:d}

>>> "my name is {:s} and my age is {:d} years".format(name, age)

'my name is Malhar and my age is 23 years'

>>>

Precision formatting of numbers can be accordingly done.

>>> "my name is {:s}, age {:d} and I have scored {:6.3f} percent  
 marks".format(name, age, percentage)

'my name is Malhar, age 23 and I have scored 55.500 percent marks'

>>>

**Other Methods for String Processing**

To learn about other methods that can be used to process strings, click each on the methods below.

* [capitalize ()](https://trainings.internshala.com/content/python#item1)
* [upper ()](https://trainings.internshala.com/content/python#item2)
* [lower()](https://trainings.internshala.com/content/python#item3)
* [title()](https://trainings.internshala.com/content/python#item4)
* [find()](https://trainings.internshala.com/content/python#item5)
* [index()](https://trainings.internshala.com/content/python#item6)
* [count()](https://trainings.internshala.com/content/python#item7)
* [isalpha()](https://trainings.internshala.com/content/python#item8)
* [isdigit()](https://trainings.internshala.com/content/python#item9)
* [islower()](https://trainings.internshala.com/content/python#item10)
* [isupper()](https://trainings.internshala.com/content/python#item11)
* capitalize(): This method converts the first character of a string to uppercase letter.
* >>> var='internshala'
* >>> var.capitalize()
* 'Internshala'
* >>>
* upper(): This method returns a string with lowercase characters replaced by corresponding uppercase characters.
* >>> var='internshala'
* >>> var.upper()
* 'INTERNSHALA'
* >>>
* lower(): This method results in a string with uppercase characters replaced by corresponding lowercase characters.
* >>> var='INTERNSHALA'
* >>> var.lower()
* 'internshala'
* >>>
* title(): This method results in a string with the first character of each word converted to uppercase.
* >>> var='python training from internshala'
* >>> var.title()
* 'Python Training From Internshala'
* >>>
* find(): This method finds the first occurrence of a substring in another string. If not found, the method returns -1.
* >>> var='python training from internshala'
* >>> var.find('in')
* 10
* >>> var.find('on')
* 4
* >>> var.find('run')
* -1
* >>>
* The substring 'in' first occurs at the 10th position (count starts from 0), 'on' is found at the 4th position, but 'run' is not found hence returns -1.
* index(): This method is similar to find() but throws a ValueError if the substring is not found.
* >>> var='python training from internshala'
* >>> var.index('in')
* 10
* >>> var.index('run')
* Traceback (most recent call last):
* File "<pyshell#19>", line 1, in <module>
* var.index('run')
* ValueError: substring not found
* >>>
* count(): This method returns the number of occurrences of a substring in given string.
* >>> var='python training from internshala'
* >>> var.count('in')
* 3
* >>>
* isalpha(): This method returns true if all the characters in a string are alphabetic (a-z or A-Z), otherwise returns false.
* >>> var='Internshala'
* >>> var.isalpha()
* True
* >>> var='Intern shala'
* >>> var.isalpha()
* False
* >>>
* isdigit(): This method returns true if all characters in a string are digits( 0-9), if not returns false.
* >>> var='2000'
* >>> var.isdigit()
* True
* >>> var='2,000'
* >>> var.isdigit()
* False
* >>>
* islower(): This method returns true if all characters in a string are lowercase characters else returns false.
* >>> var='internshala'
* >>> var.islower()
* True
* >>> var='Internshala'
* >>> var.islower()
* False
* >>> var='intern shala'
* >>> var.islower()
* True
* >>>
* isupper(): This method returns true if all characters in a string are uppercase characters else returns false.
* >>> var='INTERN\_SHALA'
* >>> var.isupper()
* True
* >>> var='INTERNshala'
* >>> var.isupper()
* False
* >>> var='INTERN+SHALA'
* >>> var.isupper()
* True
* >>> var='1234'
* >>> var.isupper()
* False
* >>>

#### **Set Data type**

Set is also a collection data type in Python. However, it is not an ordered collection of objects, like list or tuple. Hence, indexing and slicing operations cannot be done on a set object. A set also doesn’t allow duplicate objects to be stored, where as in list and tuple, the same object can appear more than once. Even if an object is put more than once in a set, only one copy is held. Set is a Python implementation of a set as defined in Mathematics. The set object has suitable methods to perform mathematical set operations like union, intersection, difference etc. A set object contains one or more items, not necessarily of the same type which are separated by comma and enclosed in curly brackets {}.

>>> S1={1, "Ravi", 75.50}

>>> S1

{1, 75.5, 'Ravi'}

>>> type(S1)

<class 'set'>

>>> S2={10,23,40,23,50,10}

>>> S2

{40, 10, 50, 23}

>>>

##### **set() function**

Python has an in-built function set() using which set object can be constructed out of any sequence like string, list or tuple object.

>>> S1=set("Internshala")

>>> S1

{'t', 'n', 's', 'h', 'e', 'a', 'l', 'I', 'r'}

>>> S2=set([45,67,87,36,55])

>>> S2

{55, 67, 36, 45, 87}

>>> S3=set((10,25,15))

>>> S3

{25, 10, 15}

>>>

Order of elements in the set is not necessarily the same that is given at the time of assignment. Python optimizes the structure for performing operations over set as defined in mathematics. Only immutable (and hashable) objects can be a part of set object. Numbers (integer, float as well as complex), strings, and tuple objects are accepted but list and dictionary objects are not.

>>> S1={(10,10), 10,20}

>>> S1

{10, 20, (10, 10)}

>>> S2={[10,10], 10,20}

Traceback (most recent call last):

File "<pyshell#2>", line 1, in <module>

S2={[10,10], 10,20}

TypeError: unhashable type: 'list'

>>>

In first case, (10,10) is a tuple, hence it becomes part of set. In second example though, since [10,10] is a list, error message is displayed saying list is unhashable. (Hashing is a mechanism in computer science which enables quicker searching of objects in computer’s memory. <https://en.wikipedia.org/wiki/Hash_function>) Even though mutable objects are not stored in a set, set itself is a mutable object. A set object can be modified by add(), update(), remove() and discard() methods. 

* [add()](https://trainings.internshala.com/content/python#item1)
* [update()](https://trainings.internshala.com/content/python#item2)
* [clear()](https://trainings.internshala.com/content/python#item3)
* [copy()](https://trainings.internshala.com/content/python#item4)
* [discard()](https://trainings.internshala.com/content/python#item5)
* [remove()](https://trainings.internshala.com/content/python#item6)

##### **Set Operations**

As mentioned earlier, set data type in Python implements set as defined in mathematics. Various Set operations can be performed using Python’s set obect. The operators |, &, - and ^ perform union, intersection, difference and symmetric difference operations respectively. Each of these operators have a corresponding method associated with built-in set class.

* [Union](https://trainings.internshala.com/content/python#item7)
* [Intersection](https://trainings.internshala.com/content/python#item8)
* [Difference](https://trainings.internshala.com/content/python#item9)
* [Symmetric Difference](https://trainings.internshala.com/content/python#item10)

Set is a specialized data type. One of the major applications of Python is in area of mathematical computing and data analysis in which set operations are important. We may drop this discussion considering it as not for beginner (and also to curtail the size), but learner (especially who intends to go in mathematical and scientific computing) should be encouraged to explore this section. We should emphasize this and provide this as a text for further reading.

Concatenation

Appends the second list or tuple to the first.  
  
The data types being concatenated should be of the same type. For example, you cannot concatenate a list and a tuple.   
  
List

>>> founders=["Iron Man", "Thor", "Ant-Man", "Hulk", "Wasp"]

>>> recruits=["Captain America", "Hawkeye"]

>>> founders + recruits

['Iron Man', 'Thor', 'Ant-Man', 'Hulk', 'Wasp', 'Captain America', 'Hawkeye']

>>>

Tuple

>>> stark=("Ned", "Catelyn", "Brandon", "Sansa", "Arya", "Robb")

>>> mormont=("Jeor", "Maege", "Jorah", "Lyanna")

>>> stark + mormont

('Ned', 'Catelyn', 'Brandon', 'Sansa', 'Arya', 'Robb', 'Jeor', 'Maege', 'Jorah', 'Lyanna')

>>>

Repition

Concatenates multiple copies of the same list or tuple.   
  
List

>>> dna=["A", "G","T","C"]

>>> dna \* 3

['A', 'G', 'T', 'C', 'A', 'G', 'T', 'C', 'A', 'G', 'T', 'C']

>>>

Tuple

>>> glucose=("C6", "H12", "O6")

>>> glucose\*6

('C6', 'H12', 'O6', 'C6', 'H12', 'O6', 'C6', 'H12', 'O6', 'C6', 'H12', 'O6', 'C6', 'H12', 'O6', 'C6', 'H12', 'O6')

>>>

Slice

Returns the item at given index in a list or tuple.   
A negative index counts the position from right with the count starting at -1.  
  
Note: Recall that the index starts at 0.  
  
List

>>> founders=["Iron Man", "Thor", "Ant-Man", "Hulk", "Wasp"]

>>> founders[3]

'Hulk'

>>>

>>> founders[-3]

'Ant-Man'

>>>

Tuple

>>> stark=("Ned", "Catelyn", "Brandon", "Sansa", "Arya", "Robb")

>>> stark[2]

'Brandon'

>>>

>>> stark[-1]

'Robb'

>>>

Range Sice

Fetches items in a range specified by the two index operands separated by the colon [:] symbol.   
  
If the first operand is omitted, the range starts at zero index.If the second operand is omitted, range goes upto the end of the list or tuple.   
  
List

>>> avengers=["Iron Man", "Thor", "Ant-Man", "Hulk", "Wasp", "Captain America", "Hawkeye"]

>>> avengers[2:5]

['Ant-Man', 'Hulk', 'Wasp']

>>>

>>> avengers[3:]

['Hulk', 'Wasp', 'Captain America', 'Hawkeye']

>>>

>>> avengers[:3]

['Iron Man', 'Thor', 'Ant-Man']

>>>

Tuple

>>> got=("Ned", "Catelyn", "Brandon", "Sansa", "Arya", "Robb", "Jeor", "Maege", "Jorah", "Lyanna")

>>> got[1:4]

('Catelyn', 'Brandon', 'Sansa')

>>>

>>> got[3:]

('Sansa', 'Arya', 'Robb', 'Jeor', 'Maege', 'Jorah', 'Lyanna')

>>> got[4:]

('Arya', 'Robb', 'Jeor', 'Maege', 'Jorah', 'Lyanna')

>>>

In membership

Returns true if an object exists in the given list or tuple.   
  
Note: If the list or tuple comprises string items, then the items are case sensitive.   
  
List

>>> avengers=["Iron Man", "Thor", "Ant-Man", "Hulk", "Wasp", "Captain America", "Hawkeye"]

>>> "Wasp" in avengers

True

>>>

>>> "Black Widow" in avengers

False

>>>

Tuple

>>> got=("Ned", "Catelyn", "Brandon", "Sansa", "Arya", "Robb", "Jeor", "Maege", "Jorah", "Lyanna")

>>> "Arya" in got

True

>>>

>>> "Jon" in got

False

>>>

Not in membership

Returns true if an object does not exist in the given list or tuple   
  
List

>>> avengers=["Iron Man", "Thor", "Ant-Man", "Hulk", "Wasp", "Captain America", "Hawkeye"]

>>> "Wasp" not in avengers

False

>>>

>>> "Black Widow" not in avengers

True

>>>

Tuple

>>> got=("Ned", "Catelyn", "Brandon", "Sansa", "Arya", "Robb", "Jeor", "Maege", "Jorah", "Lyanna")

>>> "Arya" not in got

False

>>>

>>> "Jon" not in got

True

>>>

#### **Manipulating Lists**

You can modify the items within a list. Modifying a list means tochange an item, or add a new item, or remove an existing item. Here are some methods of the built-in List class that help in modifying lists. Read through each function and then try it out in IDLE.

* [append()](https://trainings.internshala.com/content/python#item1)
* [insert()](https://trainings.internshala.com/content/python#item2)
* [remove()](https://trainings.internshala.com/content/python#item3)
* [pop()](https://trainings.internshala.com/content/python#item4)
* [reverse()](https://trainings.internshala.com/content/python#item5)
* [sort()](https://trainings.internshala.com/content/python#item6)

Adds an item at the end of the list.

>>> L2=['Tailor Swift', 'Ed Sheeran', 'Imagine Dragons', 'Pink', 'Maroon 5']

>>> L2.append('Halsey')

>>> L2

['Tailor Swift', 'Ed Sheeran', 'Imagine Dragons', 'Pink', 'Maroon 5', 'Halsey']

>>>

Inserts an item in list at the specified index.

>>> L2=['Tailor Swift', 'Ed Sheeran', 'Imagine Dragons', 'Pink', 'Maroon 5']

>>> L2.insert(2, 'Halsey')

>>> L2

['Tailor Swift', 'Ed Sheeran', 'Halsey', Imagine Dragons', 'Pink', 'Maroon 5']

>>>

Removes the specified item from the list.

>>> L2=['Python', 'Perl', 'Java', 'C++'

>>> L2.remove('Java')

>>> L2

['Python', 'Perl', 'C++']

>>>

Removes and returns the last object in list.

>>> L2=['Python', 'Perl', 'Java', 'C++'

>>> L2.pop()

'C++'

>>> L2

['Python', 'Perl', 'Java']

>>>

Reverses the order of items in a list.

>>> L2=['Python', 'Perl', 'Java', 'C++'

>>> L2.reverse()

>>> L2

['C++', 'Java', 'Perl', 'Python']

>>>

Rearranges items in the list according to alphabetical order. Default is ascending order. For descending order put reverse=True as argument in function bracket.

>>> L2=['Python', 'C++', 'Java', 'Ruby'

>>> L2.sort()

>>> L2

['C++', 'Java', 'Python', 'Ruby']

>>>

>>> L2.sort(reverse=True)

>>> L2

['Ruby', 'Python', 'Java', 'C++']

>>>

#### **Converting One Sequence Type to Another**

The Following utility functions help in converting one sequence data type to other.

* [list()](https://trainings.internshala.com/content/python#item7)
* [tuple()](https://trainings.internshala.com/content/python#item8)

>>> t2=('python', 'java', 'c++')

>>> list(t2)

['python', 'java', 'c++']

>>> s1="Internshala"

>>> list(s1)

['I', 'n', 't', 'e', 'r', 'n', 's', 'h', 'a', 'l', 'a']

>>>

Converts a list or string to a tuple.

>>> L2=['C++', 'Java', 'Python', 'Ruby']

>>> tuple(L2)

('C++', 'Java', 'Python', 'Ruby')

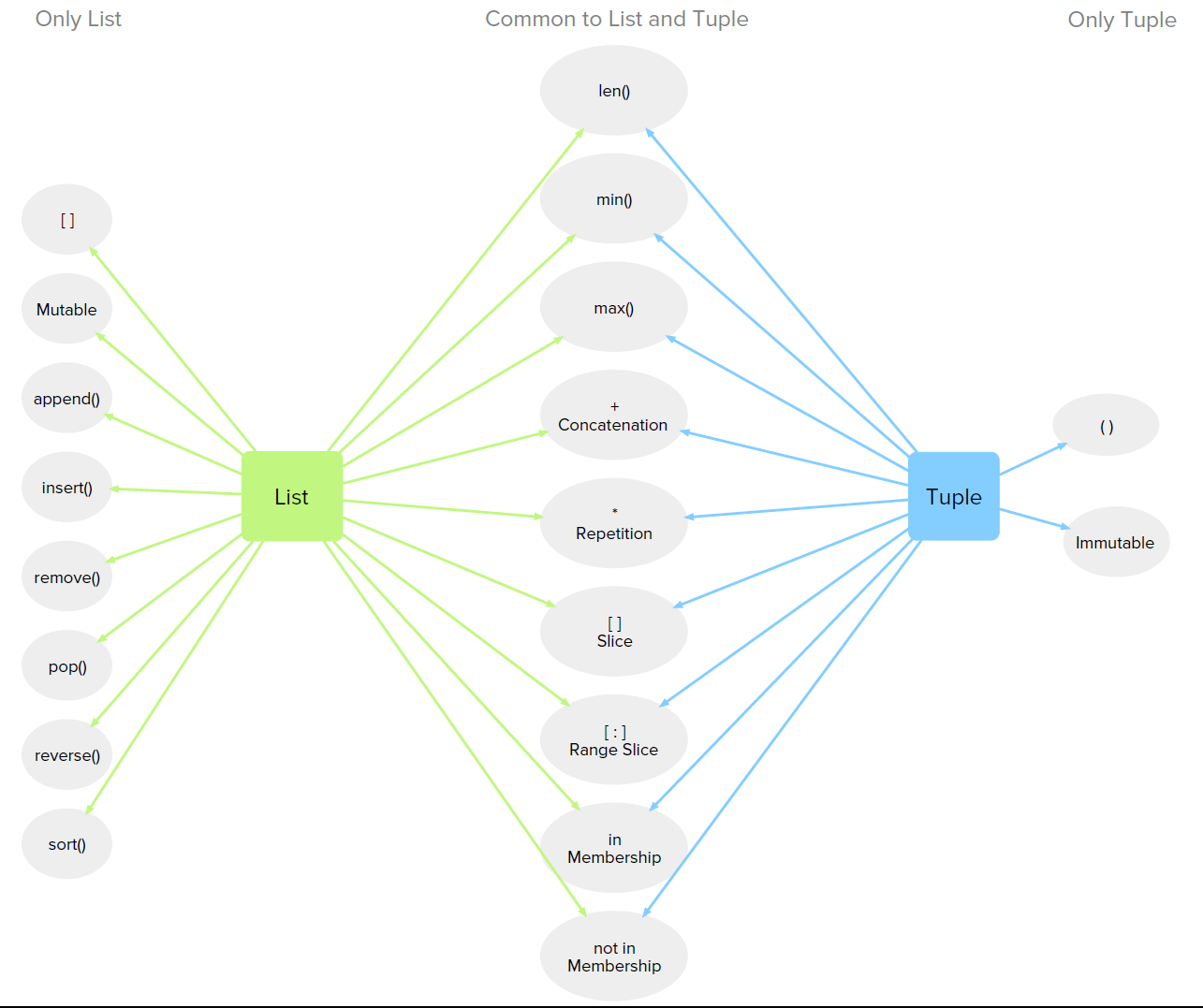
>>> s1="Internshala"

>>> tuple(s1)

('I', 'n', 't', 'e', 'r', 'n', 's', 'h', 'a', 'l', 'a')

>>>

Here's a summary of the comparison between a list and a tuple.



Len(), min() and max() are also used in dictionary

Del deciton\_name[key] deletes the key and its value

If dedict\_name , it delets the whole dict

##### **Other Methods Of Dictionary Objects**

* [items()](https://trainings.internshala.com/content/python#item1)
* [keys()](https://trainings.internshala.com/content/python#item2)
* [values()](https://trainings.internshala.com/content/python#item3)
* This method returns a list of tuples, with each tuple containing one key and and the
* corresponding value.
* >>> captains={'England': 'Root', 'Australia': 'Smith', 'India': 'Virat', 'Pakistan': 'Sarfraz'}
* >>> captains.items()
* dict\_items([('England', 'Root'), ('Australia', 'Smith'), ('India', 'Virat'), ('Pakistan', 'Sarfraz')])
* >>>

This method returns a list object comprising keys in the dictionary. 

>>> captains={'England': 'Root', 'Australia': 'Smith', 'India': 'Virat', 'Pakistan': 'Sarfraz'}

>>> captains.keys()

dict\_items(['England', 'Australia', 'India', 'Pakistan'])

>>>

This method returns a list object comprising values in the dictionary. 

>>> captains={'England': 'Root', 'Australia': 'Smith', 'India': 'Virat', 'Pakistan': 'Sarfraz'}

>>> captains.values()

dict\_items([Root', 'Smith', 'Virat', 'Sarfraz'])

>>>

1. What is the correct expression for calculating the cube root of 27?

(1 Mark)

1. 27\*\*(1/3)
2. 27\*\*1/3
3. 27//(1/3)

2. What will be result of following expression?   
  
'INTERNSHALA'.lower().isupper()

(2 Marks)

1. Result will be, 'True'.
2. Result will be, 'False'.

3. What is the value of following arithmetic expression:   
  
5\*2\*\*10

(3 marks)

1. 100
2. 10000000000
3. 5120

4. Is this statement true or false?  
  
The variable name, \_books is a valid name.

(1 Mark)

1. True
2. False

5. Which of the following statements is true about the functions, index() and find()?

(2 Marks)

1. Both give the position of the first occurrence of the substring.
2. While index() returns the first occurrence of the substring, find() returns the last occurrence.
3. Both functions return the number of times a substring appears.

6. Given the following assignments, what is the value of the variable, b?   
  
>>> a=4  
>>> b=a  
>>> a="Internshala"  
>>> b

(3 marks)

1. 4
2. Internshala
3. a
4. You will get an error

7. Is this statement true or false?  
  
The variable name, 3score is a valid name.

(1 Mark)

1. True
2. False

8. What is result of the following expression:  
  
>>> max('ten', 'twenty', 'thirty')

(2 Marks)

1. ten
2. twenty
3. thirty

9. Which of the following conversions can be done using the int() function?

(3 marks)

1. Integer to Float
2. Hexadecimal to integer
3. Integer to binary

Correct Answer : b

10. What is the index of the item ' learn salsa' in the list:   
  
bucketlist=[" Get a tattoo", "Try a new hair colour", "share a meal with a stranger", "learn salsa", "learn pottery"]

(1 Mark)

1. 5
2. 4
3. 3
4. 2

11. Which of the following is an invalid expression?

(2 Marks)

1. [1,2]+[4]
2. [1,2]\*4
3. (1,2)\*4
4. [1,2]^4

12. What is the purpose of the function, pop()?

(3 marks)

1. Remove the first element from the list.
2. Remove the last element from the list.
3. Highlight the first and last elements in the list.
4. Remove all elements from the list.

13. Which of the following statements best describes a dictionary item in Python?

(1 Mark)

1. A dictionary is a sequence of key-value pairs
2. A dictionary is an indexed collection of key-value pairs.
3. A dictionary is an unordered collection of key-value pairs

14. What is the scientific notation for the float number, 0.0000001234?

(2 Marks)

1. 1234e-10
2. 1.234e-07
3. 0.1234E-6

15. Consider the dictionary, years={1995:'Java', 1972:'C', 1994:'Python'}. Will both the following expressions return the same result?   
  
>>>min(years)   
>>>min(years.keys())

(3 marks)

1. Yes, both expressions will return the same result.
2. No, the result from both expressions will be different.

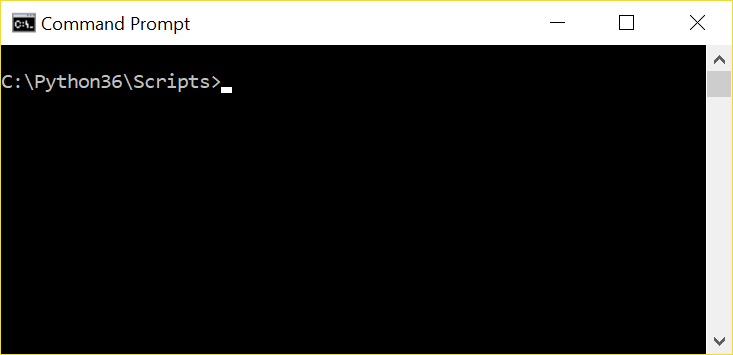
Converting Python scripts into executable programs - Windows

py2exe is a Python extension which converts Python scripts (.py) into Microsoft Windows executables (.exe). These executables can run on a system without Python installed.

Installing py2exe

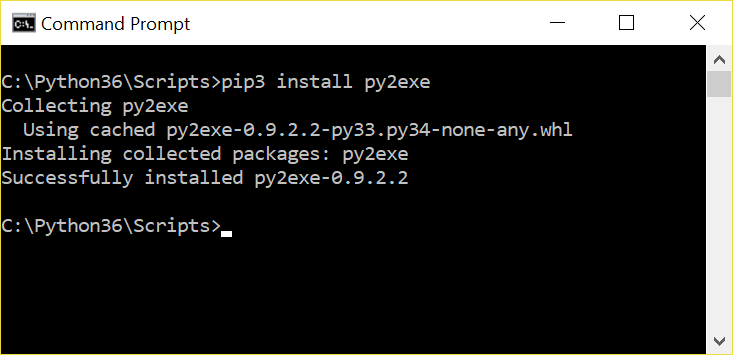
For Windows systems, you can download a third-party tool called 'py2exe'.

* 1. Run the Command Prompt.
  2. Change to the Python\Scripts directory.



1. Download py2exe using the built-in utility pip:

|  |
| --- |
| pip3 install py2exe |



Converting a script into an executable

1. For each script file that needs to be converted into an executable, a setup script file needs to be created. Assuming that test.py is to be converted into an executable, a setup.py file should be created with the following lines of code.

|  |
| --- |
| from distutils.core import setup     import py2exe     setup(console=['test.py']) |

1. Run this script from command-line

|  |
| --- |
| python setup.py py2exe |

1. Two directories will be created when you run your setup script, build and dist. The test.exe will be stored in dist directory.

|  |
| --- |
| C:\test>cd dist     C:\test\dist>test.exe |

Converting Python scripts into executable programs - Linux

Using chmod in Linux

1. Open terminal window from Linux GUI.

The chmod command changes file attributes. Every file has read, write and execute attributes. Any attribute can be set or unset by + or – operator. To make test.py an executable use following command

|  |
| --- |
| $ chmod +x test.py |

You no longer need to execute test.py with Python interpreter. Instead test executable script is run by followingcommand

|  |
| --- |
| $./test |

Passing Arguments by reference

A variable is Python is a reference to the object memory. So, both formal and actual arguments refer to the same object. The following code snippet will confirm this.

We have used the id() function earlier. It returns a unique integer corresponding to identity of an object. In the code displayed above, the id() of a list object before and after passing to a function shows an identical value.

>>> def myfunction(newlist):

print("List accessed in function: ", "id:", id(newlist))

return

>>> mylist=[10,20,30,40,50]

>>> print("List before passing to function: ", "id:", id(mylist))

List before passing to function: id: 2430484894856

>>> myfunction(mylist)

List accessed in function: id: 2430484894856

>>>

If we modify the list object inside the function and display its contents after the function is completed, changes are reflected outside the function as well.

The following result confirms that arguments are passed by reference to a Python function.

>>> def myfunction(list):

list.append(60)

print("modified list inside function:", list)

return

>>> mylist=[10,20,30,40,50]

>>> print("list before passing to function:", mylist)

list before passing to function: [10, 20, 30, 40, 50]

>>> myfunction(mylist)

modified list inside function: [10, 20, 30, 40, 50, 60]

>>> print("list after passing to function:", mylist)

list after passing to function: [10, 20, 30, 40, 50, 60]

>>>

MODULES

* [\_\_builtin\_\_](https://trainings.internshala.com/content/python#item1)
* [exceptions](https://trainings.internshala.com/content/python#item2)
* [os](https://trainings.internshala.com/content/python#item3)
* [string](https://trainings.internshala.com/content/python#item4)
* [re](https://trainings.internshala.com/content/python#item5)
* [math](https://trainings.internshala.com/content/python#item6)
* [cmath](https://trainings.internshala.com/content/python#item7)
* [sys](https://trainings.internshala.com/content/python#item8)
* [time](https://trainings.internshala.com/content/python#item9)
* [gc](https://trainings.internshala.com/content/python#item10)
* [\_\_builtin\_\_](https://trainings.internshala.com/content/python#item1)

This module contains built-in functions which are automatically available in all Python modules. You usually don’t have to import this module; it is loaded automatically when the interpreter starts.

* [exceptions](https://trainings.internshala.com/content/python#item2)
* This module provides the standard exception hierarchy. It is automatically imported when Python starts.   
  An exception is an eventthat occurs during the execution of a program and disrupts the normal flow of the program's instructions. An exception is a Python object that represents an error.

Os

This module provides a unified interface to a number of operating system functions.

String

This module contains a number of functions for string processing.

Re

This module provides a set of powerful regular expression facilities. A Regular Expression (RegEx) allows powerful string search and matching for a pattern in a string.

Math

This module implements a number of mathematical operations for floating point numbers. These functions are generally thin wrappers around the platform C library functions.

Cmath

This module contains a number of mathematical operations for complex numbers.

Sys

This module provides functions and variables used to manipulate different parts of the Python runtime environment.

Time

This module provides functions to deal with dates and the time within a day. It wraps the C runtime library.

Gc

This module provides an interface to the built-in garbage collector.

**OS**

**Functions in this module are:**

mkdir()-new dirctory

chdir()-change directory

rmdir()-remove directory

getcwd()-current working directory

listdir()-list content of present directory

os.mkdir(“d:\\fhsdnkml”)

os.chdir(“d:\\ihgfihd”)

os.getcwd(“d:\\gnjdfk”)

**RANDOM**

**Functions in this module are:**

Random

Randint

randrange

Choice

Shuffle

Random.random()

Random.randint(start,stop,)

Random.randrange(start,stop,step)

Random.choice(“any string,list”)

Random.shuffle(“list”)

**MATH**

**Functions in this module are:**

Math.pi

Math.e

Sin()

Cos()

Tan()

Degrees()

Radians()

Log()

Log10()

Exp()

Pow()

Sqrt(base, power)

Ceil() - round off to smallest integer[example-4.8to5]

Floor()- round off to largest integer[example-4.8to4]

Reloading a module

Let us assume the test.py module contains the following function defined in it.

def SayHello(name):

print("Hi {}! How are you?".format(name))

return

We call SayHello() function after importing the module, test.

>>> import test

>>> test.SayHello("Deepak")

Hi Deepak! How are you?

>>>

Now assume we need to modify the function SayHello()before executing it again. The updated function is:

def SayHello(name,course):

print("Hi {}! How are you?".format(name))

print("Welcome to {} online training program by Internshala".format(course))

return

In order to call the SayHello() function without ending the current interpreter session, we can reload it by using reload() function from imp module

>>> import test

>>> test.SayHello("Deepak")

Hi Deepak! How are you?

>>> import imp

>>> imp.reload(test)

<module 'test' from 'C:\\Python\\test.py'>

>>> test.SayHello("Deepak", "Python")

Hi Deepak! How are you?

Welcome to Python online training program by Internshala

>>>

**MODULE ATTRIBUTES**

\_dict\_ returns the list of all attributes of the module

\_file\_ returns the path of the module

\_doc\_ returns the comments in the top of file

\_name\_ returns the name of the module

Dir(module name) returns the list of functions in the module

Using “from” we can import specific functions from the module

From friends import movie

Creating a Package Setup File

from setuptools import setup

setup(name='MyFriends', version='1.0', description='A package to calculate expenses when you hang-out with friends.',

url='#',

author='internshala',

author\_email='internshala@internshala.com',

license='MIT',

packages=['MyFriends'],

zip\_safe=False)

Here is a list of magic methods used to overload different operators.

|  |  |
| --- | --- |
| Operator | Method |
| + | object.\_\_add\_\_(self, other) |
| - | object.\_\_sub\_\_(self, other) |
| \* | object.\_\_mul\_\_(self, other) |
| // | object.\_\_floordiv\_\_(self, other) |
| / | object.\_\_div\_\_(self, other) |
| % | object.\_\_mod\_\_(self, other) |
| \*\* | object.\_\_pow\_\_(self, other[, modulo]) |
| < | object.\_\_lt\_\_(self, other) |
| <= | object.\_\_le\_\_(self, other) |
| == | object.\_\_eq\_\_(self, other) |
| != | object.\_\_ne\_\_(self, other) |
| >= | object.\_\_ge\_\_(self, other) |

1. Which of the following is the correct syntax of inheritance?

(1 Mark)

1. class derived(class base):
2. class derived(base):
3. def class derived(base):
4. None of the above

2. What is self?

(2 Marks)

1. It is a variable.
2. It is a class attribute.
3. It is reference to object which calls an instance method.
4. It is a built-in function in python

Correct Answer : c

3. Which library function shows list of magic methods inherited from object class?

(3 marks)

1. str()
2. dir()
3. magic()
4. None of the above

4. To overload an operator corresponding magic method should be

(1 Mark)

1. Inherited
2. Overridden
3. Overloaded

5. What is not true about a getter method?

(2 Marks)

1. Retrieves value of instance variable
2. Prints value of instance variable
3. Its name starts with ‘get’
4. It is an instance method

Correct Answer : c

6. What is not true about overriding in Python?

(3 marks)

1. Redefining a base class method in inherited class is called method overriding
2. Overriding is essential feature of object oriented language.
3. Overridden method must have same number of arguments as base class method of same name.
4. All the above statements are true.

7. Mechanism of designing a new class based on one or more existing classes is called

(1 Mark)

1. Overriding
2. Inheritance
3. Polymorphism
4. None of these

Correct Answer : b

8. What is not true about overriding?

(2 Marks)

1. Overriding can occur in case of inheritance of classes.
2. It is a process of introducing new method in inherited class
3. It is a process of redefining inherited method in child class
4. It is not mandatory to override a base class method in inherited class

Correct Answer : b

9. class book:  
    def \_\_init\_\_(self):  
        self.name="Python Programming"  
        self.price=200  
b1=book()  
  
Based on the above, which of the following statements is correct?

(3 marks)

1. print (b1.price)
2. print (book.\_\_price)
3. print(\_book.\_\_price)
4. print(b1.\_book\_\_price)

10. Which of the following is a feature of object oriented programming?

(1 Mark)

1. Data encapsulation
2. Inheritance
3. Polymorphism
4. All of the above

11. Choose incorrect option.

(2 Marks)

1. Class method is defined with @classmethod decorator
2. Class method is defined inside a class
3. Class method can access only class attributes
4. Class method has one argument called self

12. What is not true about class attribute?

(3 marks)

1. It is defined in \_\_init\_\_() method.
2. Its definition is not present inside any method
3. It is accessible by name of class
4. All options above are true

13. Which of the following is not a procedure oriented language?

(1 Mark)

1. C
2. FORTRAN
3. Java
4. PASCAL

Correct Answer : c

14. A class may contain

(2 Marks)

1. Instance attributes and instance methods
2. class attributes and class methods
3. only a
4. both a and b

15. The \_\_add\_\_() method

(3 marks)

1. Returns addition of two numbers
2. Overloads + operator
3. Should be overridden to overload + operator
4. None of the above

SQLite Data Types

Let's talk a little about data types that are used in SQLite databases. SQLite uses what is called, dynamic type system. That means, the value stored in a column determines its data type and not the column's data type. Also, you don’t have to define a specific data type for a column when you create a table. Even if you have a column with the integer data type for example, you can store any kind of data types such as text and SQLite will not complain about this.

The ANSI Standard of SQL specifies the data types to be used by relational databases. SQLite provides the following five data types which are referred to as storage classes:

|  |  |
| --- | --- |
| Storage Class | Meaning |
| NULL | Missing or unknown information. |
| INTEGER | Whole numbers, either positive or negative. |
| REAL | Real numbers with decimal values that use 8-byte floats. |
| TEXT | Character data. SQLite supports various character encodings. |
| BLOB | Binary Large Object that can be used to store any kind of data. The maximum size of BLOBs is unlimited. |

A storage class is more general than a datatype. These storage classes are mapped to standard SQL data types. For example, INTEGER in SQLite has a type affinity with all integer types such as int, smallint, bigint, tinyint etc. Similarly REAL in SQLite has a type affinity with float and double data type. Standard SQL data types such as varchar, char, nchar etc. are equivalent to TEXT in SQLite.

Inserting a new Record

In the previous section, you learned how to create a database and a table within it. Now, let's see how to create a new record in the existing table, student.

Example 1: To add the following record:

    Name = Sherlock

    House = Slytherin

    Marks = 65

1. Assuming that the database MySchool is created and contains the table student, we start by creating a connection:

import sqlite3

MySchool=sqlite3.connect('schooltest.db')

curschool=MySchool.cursor()

2. To add a new record to the table, we execute the INSERT query.

import sqlite3

MySchool=sqlite3.connect('schooltest.db')

curschool=MySchool.cursor()

curschool.execute("INSERT INTO student (StudentID, name, house, marks) VALUES (5,'Sherlock',32,50);")

3. We now commit the changes to confirm them.

import sqlite3

MySchool=sqlite3.connect('schooltest.db')

curschool=MySchool.cursor()

curschool.execute("INSERT INTO student (StudentID, name, house, marks) VALUES (5,'Sherlock',32,50);")

MySchool.commit()

The new record is added to the table. You can verify this from SQLite Studio.  
You can refer to the helper text for the relevant screenshot.

Inserting a new Record

Example 2: To accept user input for the values in the table:

1. Instead of adding known values, you can also accept user input for these values. Assuming that the database MySchool is created and contains the table student, we start by creating a connection:

import sqlite3

MySchool=sqlite3.connect('schooltest.db')

curschool=MySchool.cursor()

2. To accept user input, we use variables to store each of the values.

import sqlite3

MySchool=sqlite3.connect('schooltest.db')

curschool=MySchool.cursor()

mysid= int(input("Enter ID: "))

myname=input("Enter name: ")

myhouse=int(input("Enter house: "))

mymarks=float(input("Enter marks: "))

3. We now replaces the fixed VALUES in the INSERT query with the variables, mysid, myname, myhouse and mymarks. To do this, we use the DB-API’s parameter substitution. We put a ? as a placeholder wherever we want to use a value and then give a tuple of values as the second argument to the cursor’s execute() method.

import sqlite3

MySchool=sqlite3.connect('schooltest.db')

curschool=MySchool.cursor()

mysid= int(input("Enter ID: "))

myname=input("Enter name: ")

myhouse=int(input("Enter house: "))

mymarks=float(input("Enter marks: "))

curschool.execute("INSERT INTO student (StudentID, name, house, marks) VALUES (?,?,?,?);", (mysid,myname,myhouse,mymarks))

4. We now commit the changes.

import sqlite3

MySchool=sqlite3.connect('schooltest.db')

curschool=MySchool.cursor()

mysid= int(input("Enter ID: "))

myname=input("Enter name: ")

myhouse=int(input("Enter house: "))

mymarks=float(input("Enter marks: "))

curschool.execute("INSERT INTO student (StudentID, name, house, marks) VALUES (?,?,?,?);", (mysid,myname,myhouse,mymarks))

MySchool.commit()

Error handling in Python is done through the use of exceptions that are caught in try blocks and handled in except blocks.  
Let us look an example of how this is used. The following code not only accepts a user input and adds a new record but also displays a message if the operation was successful or not.

import sqlite3

MySchool=sqlite3.connect('schooltest.db')

mysid= int(input("Enter ID: "))

myname=input("Enter name: ")

myhouse=int(input("Enter house: "))

mymarks=float(input("Enter marks: "))

#try block to catch exceptions

try:

curschool=MySchool.cursor()

curschool.execute("INSERT INTO student (StudentID, name, house, marks) VALUES (?,?,?,?)", (mysid, myname, myhouse, mymarks))

MySchool.commit()

print ("One record added successfully.")

#except block to handle exceptions

except:

print ("Error in operation.")

MySchool.rollback()

MySchool.close()

The connection class defines the commit() and rollback() methods. Changes in database are finalised only if the execute() method runs successfully by commit() method. Otherwise, any changes are undone by the rollback() method. You can try this yourself by saving this code as a .py file and executing it.

#### **Try and Except**

The try statement works as follows.

* First, the try clause (the statement(s) between the try and except keywords) is executed.
* If no exception occurs, the except clause is skipped and execution of the try statement is finished
* If an exception occurs during execution of the try clause, the rest of the clause is skipped. Then the except clause is executed, and then execution continues after the try statement.
* If an exception occurs which does not match the exception named in the except clause, it is passed on to outer try statements; if no handler is found, it is an unhandled exception and execution stops with a message.

Further Reading: <https://docs.python.org/3/tutorial/errors.html>

fetchone()

The SELECT query forms a result set containing all records returned as a response to a query. The execute() method uses a string representing the SELECT query statement. There are two prominent methods as per DB-API standard. The below two methods are used:

This method fetches the next available record from the result set. It is a tuple consisting of values of each column of the fetched record. The Following code snippet retrieves and prints one record at a time till the result set is exhausted.

import sqlite3

MySchool=sqlite3.connect('schooltest.db')

sql="SELECT \* from student;"

curschool=MySchool.cursor()

curschool.execute(sql)

while True:

record=curschool.fetchone()

if record==None:

break

print (record)

fetchall()

This method fetches all the remaining records in the form of a list of tuples. Each tuple corresponds to one record and contains values of each column in the table. The following code snippet fetches all records and prints them one at a time by using the 'for' statement.

import sqlite3

MySchool=sqlite3.connect('schooltest.db')

sql="SELECT \* from student;"

curschool=MySchool.cursor()

curschool.execute(sql)

result=curschool.fetchall()

for record in result:

print (record)

Updating a Record - Example

import sqlite3

MySchool=sqlite3.connect('schooltest.db')

nm=input("enter name: ")

sql="SELECT \* from student WHERE name='"+nm+"';"

curschool=MySchool.cursor()

curschool.execute(sql)

record=curschool.fetchone()

print (record)

m=float(input("enter new marks: "))

sql="UPDATE student SET marks='"+str(m)+"' WHERE name='"+nm+"';"

try:

curschool.execute(sql)

MySchool.commit()

print ("record updated successfully")

except:

print ("error in update operation")

MySchool.rollback()